FAST DIAGONAL INTERLEAVED PARITY (DIP) CALCULATOR

Shu Yuan, Thomas A. Peterson, & Kevin E. Sallese

TECHNICAL FIELD

The present invention relates to parity calculation, and more particularly to the calculation of a diagonal interleaved parity (DIP) word.

BACKGROUND

Although modern communication protocols enable the transmission of billions of bits per second, conventional backplane switching systems and related components do not have comparable clock rates. For example, the System Packet Interface 4 (SPI4) Phase 2 (SPI4-2) protocol requires a minimum throughput rate of 10 gigabits per second over a SPI4-2 native bus having a width of 16 bits using Double Data Rate (DDR) techniques. At a throughput rate of 10 gigabits, such a bus is thus sampled at a 625 MHz rate. Because of the DDR sampling (sampling at both the rising and falling edge of the clock), the bus is clocked at 312.5 MHz. However, many application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs) cannot achieve even a 312.5 MHz clocking rate. Thus, external SPI4-2 buses routed to such devices must be demultiplexed according to a slower single edge clock rate that is a fraction of the external 625 MHZ sampling rate for the native SPI4-2 bus. For example, an FPGA having a single edge clock rate that is 1/4th the sampling rate of the native SPI4-2 bus receives four 16-bit words (typically denoted as tokens) per FPGA clock cycle. The four tokens are then routed within the FPGA on a four-token wide bus that is clocked at the lower clock rate. In general, the native SPI4-2 bus is demultiplexed according to an FPGA clock that is 1/nth the rate of the bus clock, where n is a positive integer. As just discussed, using a value of n=4 is typical although that may be increased to, for example, a value of n=8 if the FPGA clock rate is relatively slow. At each cycle of the FPGA clock, n words or tokens are demultiplexed from the SPI4-2 native bus.

This demultiplexing of the native SPI4-2 bus causes a number of complications when implementing a SPI4-2 interface using a PLD such as an FPGA. For example, the SPI4-2 standard uses a diagonal interleaved parity (DIP) scheme for point-to-point error detection. In a SPI4-2 interface, a SPI4-2 packet such as packet 100 shown in Figure 1 includes a variable number of sixteen-bit data words 105 that are followed by a single sixteen-bit control word 110. Packet 100 (which may also be denoted as a SPI4-2 burst) thus does not include a control word 115 from a previously-transmitted packet. As illustrated in Figure 1, packet 100 includes eight data words 105 but it will be appreciated that the number of data words in a given SPI4-2 packet will vary depending upon the application. However, regardless of the number of data words 105 included in a SPI4-2 packet, the packet's end is demarcated by control word 110.

Having received the control word 110 for packet 100, a sixteen bit parity word 120 may be calculated using a diagonal-interleaved parity (DIP) scheme. Each bit of parity word 120 corresponds to a diagonal XOR calculation chain starting at the first data word 105 in packet 100. For example, a diagonal exclusive OR (XOR) calculation chain 121 starts from the most significant bit (bit position 15) of the first data word 105 and propagates through the remaining data words 105 and control word 110 to produce the value for bit position 7 of parity word

120. Calculation chain 121 begins with the XOR of the most significant bit of the first data word 105 and the next-most-significant bit (bit position 14) of the second data word 105. As can be seen from Figure 1, bit position 15 of the first data word 105 holds a logical one whereas bit position 14 of the second data word 105 holds a logical zero. The XOR product is thus a logical 1. This XOR product propagates through calculation chain 121 by being XORed with the bit stored in bit position 13 of the third data word 105, the resulting XOR product then XORed with the bit stored in bit position 12 of the fourth data word 105, and so on, until the final XOR product is XORed with the bit stored in bit position 7 of control word 110 to produce a value for bit position 7 of parity word 120. It may be seen that the XOR product of the resulting bit sequence {1,0,0,1,0,0,0,0,1} in calculation chain 121 produces a value of logical one for bit position 7 of parity word 120.

The remaining XOR calculation chains are processed analogously. For example, XOR calculation chain 122 starts at bit position 14 of the first data word 105 and propagates through the remaining data words 105 and control word 110. In chain 122, the starting bit is XORed with the bit stored in bit position 13 of the second data word 105. The resulting XOR product is XORed with the bit stored in bit position 12 of the third data word 105, and so on, until the value for bit position 6 of parity word 120 is produced. Note that the least four significant bits of control word 110 are replaced with logical ones during the calculation of the least four significant bits for parity word 120.

There will always be XOR calculation chains that must wrap around in a circular modulo-16-bit fashion. For example, XOR calculation chain 123 starts at bit position 2 of the first data word 105 before propagating through the remaining data words 105 and control word 110. By the third data word 105, chain 123 is at the least significant bit (bit position 0). Thus chain 123 must wrap around to the most significant bit (bit position 15) as it propagates through the fourth data word 105.

After sixteen-bit parity word 120 has been calculated, its most significant byte is XORed with the least significant byte to produce 8-bit parity word 130. In turn, parity word 130 is folded and the two halves XORed to produce a DIP4 parity word 135. In this fashion, sixteen-bit parity word 120 is collapsed to produce DIP4 parity word 135. In a receive function, DIP4 parity word 135 is compared to the original value stored in the least four significant bits of control word 110 (which had been treated as being all logical ones for the DIP calculation) to determine if the data words 105 and control word 110 were received correctly. Conversely, in a transmit function, DIP4 parity word 135 would replace these four bits in control word 110.

The calculation of DIP4 parity word 135 becomes problematic when performed by a programmable logic device such as an FPGA as a result of the demultiplexing of the native SPI4-2 bus. Because of the demultiplexing, the position of the control word cannot be readily determined, requiring in prior approaches that a number of sets of calculation chains be calculated.

As discussed above, to implement a SPI4-2 interface in an FPGA, there will be n 16-bit words from packet 100 received for every FPGA clock cycle. Should the received packet contain more than n words, the XOR calculation chains cannot be finished in just one FPGA clock cycle. For example, assume that n equals four as discussed previously and that the packet corresponds to packet 100 of Figure 1. At each FPGA clock cycle, four words from packet 100 will be received into a register 200 as shown in Figure 2. The four words stored within register 200 may be designated word 3 through word 0 according to their sequence within packet 100. For example, if this FPGA clock cycle is such that the beginning of packet 100 is captured, then word 3 corresponds to the first data word 105, word 2 corresponds to the second data word 105, word 1 corresponds to the third data word 105, and word 0 corresponds to the fourth data word 105. Given just these four words, it is clear that the XOR calculation chains such as chains 121, 122, and 123 of Figure 1 cannot be completed during this FPGA clock cycle.

Instead, diagonal XOR calculation chains 210 will be propagated through words 3, 2, 1, and 0 and the results stored such as in an inter-slice parity summing register 205. For example, an diagonal XOR calculation chain 210a begins at the most significant bit of word 3 and continues through bit position 14 of word 2 and bit position 13 of word 1 to include bit position 12 of word 0. This resulting value is then stored in bit position 12 of inter-slice parity summing register 205. Similarly, another diagonal XOR calculation chain 210b begins at bit position 14 of word 3 and continues through bit positions 13 of word 2 and bit position 12 of word 1 to include bit position 11 of word 0. This resulting value is then stored in bit position 11 of inter-slice parity summing register 205. At the next FPGA clock cycle, the values stored in inter-slice parity summing register 205 will load into the diagonal XOR calculation chains 210. But note that it will not be known where control word 110 will be placed within register 200. For example, with respect to packet 100, register 200 would contain the first four data words 105 in the initial FPGA clock cycle. At the second FPGA clock cycle, register 200 would contain the next four data words. Finally, at the third FPGA clock cycle register 200 would store control word 110. Because there were eight data words 105 preceding control word 110 in packet 100, control word 110 would be received as word 3 in register 200. However, if register 200 was processing a packet having nine data words 105, then control word 110 would be received as word 2 in register 200. It thus follows that control word 110 may be received as any one of words 3 through word 0 in register 200, depending upon the size of the packet being processed.

Because it cannot be predicted where control word 110 will end up in register 200, it cannot be predicted where a diagonal XOR calculation chain will end when register 200 contains control word 110. For example, diagonal XOR calculation chain 210 could end at any one of four extraction points 220a, 220b, 220c, and 220d, depending upon where control word 110 was received. If control word 110 is received as word 3, diagonal XOR calculation chain 210 would end at extraction point 220a. Alternatively, if control word 110 is received as word 2, diagonal XOR calculation chain 210 would end at extraction point 220b. As yet another alternative, if control word 110 is received as word 1, diagonal calculation chain 210 would end at extraction point 220c. Finally, if control word 110 is received as word 0, diagonal XOR calculation chain 210 would end at extraction point 220d. In this fashion, the number of XOR calculation chains is increased by n times because each extraction point must be considered. For example, with respect to a value of n=4 such as used in register 200, there would thus be four sets of diagonal XOR calculation chains, each set having 16 chains corresponding to the sixteen bits for each word in packet 100. This is very inefficient because only one set will provide the DIP4 parity word 135 as determined by which position control word 110 ends up in register 200. The 16-bit value from this set of XOR calculation chains forms parity word 120, which is then collapsed to form DIP4 parity word 135 as discussed with respect to Figure 1. However, the 16-bit values from the remaining XOR calculation chain sets would be of no use with respect to forming DIP4 parity word 135. This inefficiency is worsened as the value of n increases.

Accordingly, there is a need in the art for improved DIP parity word calculation techniques.

SUMMARY

One aspect of the invention relates to a programmable device configured to calculate a diagonal interleaved parity word for a packet formed from a sequence of data words and ending in a control word, wherein the programmable device is configured to sequentially process the packet a predetermined number of words at a time. The programmable device includes a plurality of programmable blocks, one or more of the programmable blocks being configured to implement a set of XOR calculation chains, the one or more programmable blocks being configured such that the XOR calculation chains have the same length regardless of the number of data words in the packet.

Another aspect of the invention relates to a method of calculating a diagonal interleaved parity (DIP) word from a packet formed from a succession of data words ordered from a first data word to a last data word, the packet ending in a control word. The method includes the acts of successively sampling a predetermined number of ordered words from a bus, wherein the first sample starts at the first data word; for each successive sample of words, determining whether the control word is included in the sample: if the control word is not included in the sample, propagating a set of diagonal XOR calculation chains through the sample; if the control word is included in the sample, assigning the words

following the control word in the sample to logical zeroes and then propagating the set of diagonal XOR calculation chains through the sample to provide an intermediate DIP parity word. The method also includes an act of adjusting the intermediate DIP parity word according to the number or words that were assigned values of logical zeroes to provide the DIP word.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates the XOR calculation chains necessary to calculate a DIP4 parity word for a SPI4-2 packet comprised of eight data words and one control word.

Figure 2 illustrates the implementation of a DIP4 parity word calculation in a programmable logic device that demultiplexes four words from a SPI4-2 packet at each programmable logic device clock cycle.

Figure 3 is a flowchart for a DIP parity word calculation scheme according to one embodiment of the invention.

Figure 4 illustrates a 4-input AND implementation for checking a DIP parity word according to one embodiment of the invention.

Figure 5 is a block diagram of an FPGA that may be configured to implement the DIP parity word calculation scheme of Figure 3.

Use of the same reference symbols in different figures indicates similar or identical items.

DETAILED DESCRIPTION

The diagonal interleaved parity (DIP) calculation techniques disclosed herein will be described with respect to a SPI4-2 implementation, wherein each packet is comprised of sixteen-bit words such as those discussed with respect to packet 100 of Figure 1. However, it will be appreciated that the calculation techniques disclosed herein are widely applicable to any arbitrary word width. Moreover, the DIP calculation need not be performed with respect to the SPI4-2 standard, any standard needing a DIP parity word calculation would benefit from the techniques discussed herein.

As discussed with respect to Figure 2, any system implementing a SPI4-2 standard that cannot process the tokens within a SPI4-2 packet at the rate of the native SPI4-2 bus will need to demultiplex the bus at a clock rate that is 1/nth the rate of bus' rate, where n is a positive integer. A typical value for n is four but eight may also be necessary if the bus clock rate is too high with respect to the system's clock rate. In the following discussion, the system will be assumed to be a field programmable gate array (FPGA) but it will be appreciated that the DIP parity word calculation disclosed is widely applicable to any system that must demultiplex the native SPI4-2 bus during the calculation of the DIP parity word.

At each FPGA clock cycle, n sixteen-bit SPI4-2 words (typically denoted as "tokens") are demultiplexed from the native SPI4-2 bus. To avoid the inefficiencies discussed with respect to prior art DIP parity word calculation schemes, only one set of sixteen XOR calculation chains (one for each bit in the sixteen-bit words) need be used to generate DIP4 parity word 135. Thus, regardless of the value of n, the number of XOR calculation chains remains the same. This is very efficient when compared to prior art schemes that require n sets of XOR calculation chains, each set comprised of sixteen XOR calculation chains.

To enable the use of just one set of XOR calculation chains, the present invention exploits the following property of the XOR function: an XOR calculation chain will not have its value changed by propagating through additional bits, so long as those additional bits are all logical zeroes. In other words, if a XOR calculation chain has a value of logical zero and is XORed with another logical zero, the result is still logical zero. Similarly, if a XOR calculation chain has a value of logical one and is XORed with another logical zero, the result is still logical one. In formal terms, logical zero is the identity element for an XOR operation.

This property of logical zero with respect to the XOR operation may be exploited as follows. During each FPGA clock cycle, the n words received from the demultiplexing of the native SPI4-2 bus are examined. As discussed with respect to Figure 2 for register 200, these n words have an inherent order with respect to how they were carried on the native SPI4-2 bus. In other words, to acquire the set of n words for each demultiplex cycle (or equivalently, each FPGA clock cycle), first one word is received from the native SPI4-2 bus, then another, and so on, until all n words are received. For example, word 3 is the first word received with respect to register 200 of Figure 2, word 2 is the second word received, word 1 is the third word received, and word 0 is the last word received. This order should be maintained for each set of n words so that the diagonal XOR calculation chains may be formed properly. But recall that it cannot be predicted ahead of time what position control word 110 will have in this order. Instead, control word 110 may arrive as any one of the n words. Any words arriving after control word 110 have no bearing on the calculation of DIP4 parity word 135. Thus, the identity property of logical zero with respect to an XOR calculation may be exploited by assigning all words that arrive after control word 110 to comprise all logical zeroes.

For example, assume with respect to register 200 that control word 110 is received as word 1. The bits within word 0 would then be assigned to be all logical zeroes to complete the values within register 200. However, diagonal XOR calculation chains 210 continue through word 0 as described previously. Consider diagonal XOR calculation chain 210a. Because only one set of XOR calculation chains will be used, XOR calculation chain 210a need not be complicated with the possible extraction points 220a, 220b, and 220c discussed with respect to prior art applications. Instead, XOR calculation chain 210a would have just a single extraction point 220d. Having a single extraction point, the

The same extraction point 220d would be used for the remaining diagonal XOR calculation chains 210. Because it is assumed in this example that control word 110 is received as word 1 in register 200, the prior art extraction point 220c provides the correct value for sixteen-bit parity word 120. If the correct value for sixteen-bit parity word 120 is assumed to be [1100100111101001] as shown in Figure 2, these values are shifted to the right in a circular modulo-16-bit fashion by continuing to propagate the diagonal XOR calculation chains through word 0 before extraction at point 220d. This would produce a value for sixteen-bit parity word 120 as [1110010011110100]. Thus, sixteen-bit parity word 120 must then be shifted back to the left in a circular modulo-16-bit fashion to recover the correct value. Parity word 120 may then be collapsed to produce DIP4 parity word 135 as discussed previously. It will be appreciated that any SPI4-2 system requires some means such as a buffer or FIFO so that the DIP4 parity word calculation may be aligned to start with the first data word in the packet. This same means (not illustrated) may be used to buffer the data words that are assigned to logical zeroes after the control word has been received so that their original values are not lost.

The resulting DIP calculation technique may be summarized with respect to Figure 3. At step 300, n words are demultiplexed from the native SPI4-2 bus. For example, with respect to register 200, words 3 through 0 are received. Then, at step 305, the n words are examined to see if control word 110 has been received. If control word has not been received, the diagonal XOR calculation chains may be propagated through the n words in a conventional fashion and the result stored such as in inter-slice summing register 205 at step 310. If, however, the control word 110 was received, then words received after control word 110 in the set of n words are set to all logical zeroes at step 315. The diagonal XOR chains may then be propagated through the resulting n words to produce a value for 16-bit parity word 120 at step 320. At step 320, 16-bit parity word 120 is shifted to the left one bit for each word that was set to all logical zeroes in step 315. After this adjustment, 16-bit parity word 120 may be collapsed into DIP4 parity word 135 in step 330.

Although the just-described technique is very efficient with respect to having just a single extraction point for the diagonal XOR calculation chains, additional improvements may be carried out. For example, if n equals eight, 16-bit parity word 120 may have to be shifted up to 7 bit positions. Three bits are required to code for this value. But note that 16-bit parity word 120 will be collapsed into four-bit DIP4 parity word 135. Thus, these potential shifts of up to 7 bit positions will be folded into one of three possible values. For example, if 16-bit parity word 120 must be shifted to the left by one bit position, this operation is equivalent to shifting DIP4 parity word 135 to the left by one position also. Similarly, if 16-bit parity word 120 must be shifted to the left by either 2 or 3 bit positions, such operations are equivalent to shifting DIP4 parity word 135 to the left by 2 or 3 bit positions, respectively. If 16-bit parity word 120 must be shifted by four bit positions, such an operation is equivalent to shifting DIP4 parity word 135 by no bit positions. However, if 16-bit parity word 120 must be shifted by five bit positions, such an operation is equivalent to shifting DIP4 parity word 135 by one bit position. Thus, it may be summarized that the number of bit positions that 16-bit parity word 120 must be shifted by may be converted to a 2-bit value in a circular modulo-2-bit fashion. Then, rather than shift 16-bit parity word 120, DIP4 parity word 135 is shifted by the converted bit value. In this fashion, the adjustment of from 1 to seven bits is converted by ½ to just one to 3 bits, making the required logic simpler to implement.

As described so far, the DIP4 parity word 135 calculation techniques may be used for either a transmit or receive operation. In a transmit operation, DIP4 parity word 135 is calculated and then inserted into the least four significant bit positions of control word 110. The seed values of all logical ones in these bit positions are thus replaced by DIP4 parity word 135. In a receive operation, DIP4 parity word 135 would be compared to the original values of those bit positions in control word 110 to determine if the SPI4 packet had been received correctly.

The receive operation may be modified further for additional simplification. For example, rather than replace the last four bits of control words with logical ones as discussed with respect to Figure 1, the received values may be used instead. In such a case, DIP4 parity word 135 will simply equal the seed values of all logical ones if the SPI4-2 packet has been received correctly. The check of DIP4 parity word 135 may then be minimized to the use of a 4-input AND gate 405 as seen in Figure 4 rather than a comparison between a calculated and a received value. If AND gate 405 outputs a logical one, the received packet was correct. Otherwise, if AND gate 405 outputs a logical zero, the received packet contained one or more errors.

To implement the above-described technique, an FPGA need only be configured correctly and have the appropriate registers. For example, an FPGA 500 shown in Figure 5 contains a plurality of logic blocks 505. Suppose FPGA 500 is being used for a demultiplex rate of n = 4 as described with respect to Figure 2. Inter-slice summing register 205 and register 200 are not shown in FPGA 500 for ease of illustration. Logic blocks 505 would be configured with the appropriate logic to carry out the required diagonal XOR calculation chains 210. For example, with respect to the implementation of two of diagonal XOR calculation chains 210, logic blocks 505 may be configured according to the following RTL statement:

```
par_sum_reg[0] = par_sum_reg[4]^rdata[0]^rdata[17]^rdata[34]^rdata[51]
.
.
par_sum_reg[15] = par_sum_reg[3]^rdata[15]^rdata[16]^rdata[33]^rdata[50]
```

where par_sum_reg[n] represents the nth bit stored in inter-slice summing register 205, rdata[n] represents the nth bit stored in register 200, and ^ represents an XOR operation.

The above-described embodiments of the present invention are merely meant to be illustrative and not limiting. For example, although described as being implemented in an FPGA, it will be appreciated that the DIP parity calculation techniques disclosed herein are equally applicable to an ASIC implementation of SPI4-2 interface. It will thus be obvious to those skilled in the art that various changes and modifications may be made without departing from this invention in its broader aspects. Accordingly, the appended claims encompass all such changes and modifications as fall within the true spirit and scope of this invention.